DURÉE: 2h, UNE FEUILLE A5 RECTO-VERSO MANUSCRITE AUTORISÉE

REMARQUES:
- Les exercices sont indépendants et peuvent être réalisés dans l'ordre voulu.
- Dans l'implémentation d'une méthode, on pourra utiliser n'importe quelle autre méthode définie auparavant même si celle-ci n'a pas été implémentée.
- Dans toutes les implémentations que vous écrivez, pensez à respecter le guide de syntaxe pour la programmation (règles de nommage, présentation des blocs, *etc.*).

EXERCICE 1: ALGORITHMIE (4 POINTS)

1. Écrire une fonction `str2Tab(chaine)` qui convertit une chaine de caractères en un tableau d'entiers.
   - On supposera que chaque nombre est séparé par une virgule.
   - On pourra utiliser la fonction `isnumeric()` des chaînes de caractères pour s'assurer qu'aucun mot ne s'est glissé dans la chaîne.
   - Par exemple, la chaîne "42,142,loutre,37,13,857" sera convertie en [42, 142, 37, 13, 857]
2. Écrire une fonction `nbInf(tabInt, n)` qui renvoie le nombre d'entiers inférieur à n dans le tableau tabInt.
3. Écrire une fonction `isSup(tabInt, n)` qui renvoie la position du premier entier supérieur à n dans le tableau tabInt. Si aucun entier n'est supérieur à n, on renverra -1.
4. Écrire un programme principal qui :
   - déclare la chaîne de caractères "42,142,loutre,37,13,857",
   - la convertit en un tableau d'entiers tabInt et affiche le résultat,
   - demande à l'utilisateur un entier n1 puis affiche le nombre d'entiers inférieur n1 dans tabInt,
   - demande à l'utilisateur un entier n2 puis affiche la position du premier entier supérieur à n2 dans tabInt.

EXERCICE 2: POLYMORPHISME ET APPELS DE FONCTIONS (5 POINTS)

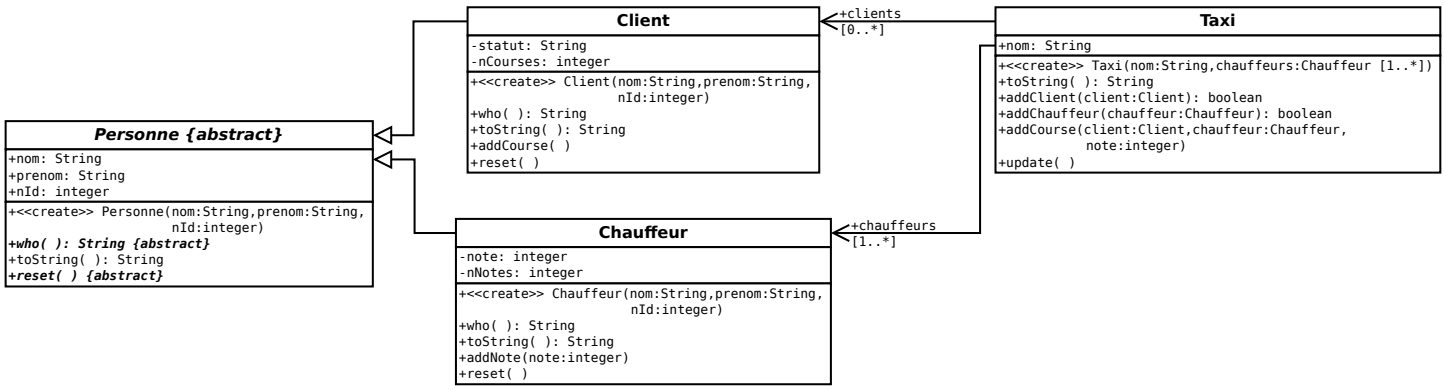
On donne ci-dessous l'implémentation de 3 classes ainsi qu'un programme principal.

```
class Un :
  def __init__( self ) :  pass
  def m1( self, other ) :
    if isinstance( other, Un ) :         print( "1-1" )
    elif isinstance( other, Deux ) :     print( "1-2" )
    elif isinstance( other, Trois ) :    print( "1-3" )

class Deux (Un) :
  def __init__( self ) : super( ).__init__( )
  def m1( other, self ) :
    if isinstance( other, Deux ):        print( "2-2" )
    elif isinstance( other, Trois ):     print( "2-3" )
    elif isinstance( other, Un ):        print( "2-1" )

class Trois (Un) :
  def __init__( self ) : super( ).__init__( )
  def m1( self, other ) :
    if isinstance( other, Trois ) :      print( "3-3" )
    if isinstance( other, Deux ) :       print( "3-2" )
    elif isinstance( other, Un ) :       print( "3-1" )

un, deux, trois = Un( ), Deux( ), Trois( )
un.m1(un),      un.m1(deux),      un.m1(trois)
deux.m1(un),    deux.m1(deux),    deux.m1(trois)
trois.m1(un),   trois.m1(deux),   trois.m1(trois)
```

1. Sachant que le code s'exécute sans erreur, quels résultats sont affichés ?

EXERCICE 3: PROGRAMMATION ORIENTÉE OBJET (11 POINTS)

On souhaite créer un logiciel de gestion de compagnie de taxis dont la modélisation UML est donnée ci-dessous. À tout moment, on pourra appeler n'importe quelle méthode présente sur le diagramme UML **même si celle-ci n'a pas encore été implémentée**. En revanche, les attributs privés n'auront **ni setters ni getters**.



1. **Classes** `Personne`**,** `Client` **et** `Chauffeur` : Une personne sera représentée par son nom, son prénom et un identifiant unique `nId`. La méthode `who( )` renverra le nom de la classe. Le statut d'un client dépendra du nombre de courses effectuées `nCourses` (Standard si moins de 5, Régulier entre 5 et 10, Premium si plus de 10). Le statut d'un client et la note d'un chauffeur pourront être réinitialiser par la méthode `reset`
   (a) Implémenter la classe `Personne` (import, attributs, méthodes). On souhaite un affichage sous la forme "Prenom Nom (Classe, nID)".
   (b) Pourquoi les méthodes `who` et `reset` sont elles abstraites ? Quelle(s) conséquence(s), cela a t'il sur la classe `Personne` ?
   (c) Implémenter la classe `Client`. On souhaite un affichage sous la forme "Prenom Nom (Classe, nID) : nCourses, Statut".
   (d) Pourquoi les attributs `statut` et `nCourses` de la classe `Client` sont-ils privés et sans setters ?
2. **Classe** `Course` : Une course sera modélisée par un client, un chauffeur et une note. Il faudra pouvoir l'afficher et lui permettre de mettre à jour les informations des clients (nCourses) et chauffeurs (note). De plus, une compagnie de `Taxi` conservera l'ensemble des courses effectuées.
   (a) Recopier le diagramme UML sur votre copie et compléter le en y ajoutant la classe `Course`. Si une classe n'est pas modifiée, vous pourrez simplement la représenter par son nom (dans un rectangle).
   (b) Justifier vos choix des visibilités des attributs de la classe Course ainsi que de ses relations avec les autres classes de l'application.
3. **Classe** `Taxi` :
   (a) Implémenter la déclaration de la classe `Taxi` ainsi que le constructeur et la méthode `toString`.
   On affichera le nom de la compagnie, le nombre de chauffeurs et de clients puis la liste des chauffeurs et clients (1 par ligne).
   (b) Implémenter la méthode `addClient` qui ajoute un `Client` à l'attribut `clients` si celui-ci n'est pas déjà présent dans la liste. On affichera un message d'erreur compréhensible si l'ajout n'est pas possible et on renverra un booléen indiquant si l'ajout a réussi ou pas.
   (c) Cette fonction nécessite t'elle de modifier la classe `Client` ou `Personne` ? Justifier votre réponse et donner l'implémentation éventuelle.
   (d) Implémenter la méthode `update` qui réinitialise les statuts des clients et les notes des chauffeurs puis parcourt l'ensemble des courses pour les remettre à jour. On supprimera l'ensemble des courses à l'issue de cette étape.